(19)日本国特許庁（JP）

(12) 公開特許公報（A）

(11)特許出願公開番号
特開2001－168733
（P2001－168733A）

(43)公開日 平成13年6月22日(2001.6.22)

(51)Int.Cl.⁷	識別記号	FI	テーマコード（参考）
H03M 13/09		H03M 13/09	
G06F 11/10	330	G06F 11/10	330S

審査請求 未請求 請求項の数4 OL （全17頁）

(21)出願番号 特願2000－312558(P2000－312558)

(22)出願日 平成12年10月12日(2000.10.12)

(31)優先権主張番号 9912710

(32)優先日 平成11年10月12日(1999.10.12)

(33)優先権主張国 フランス（FR）

(71)出願人 591000827
トムソン－セーエスエフ
THOMSON－CSF
フランス国、75008・パリ、ブルバール・オースマン・173

(72)発明者 ピエール アンドレ ローラン
フランス国 95550 ベッサンクール，シュマン デ ムニエ 114

(74)代理人 100109726
弁理士 園田 吉隆 （外1名）

(54)【発明の名称】 LDPCコードの構築およびコーディングのためのプロセス

(57)【要約】

【課題】 バイナリ情報列を保護するためのLDPCコードを簡易に構築する。

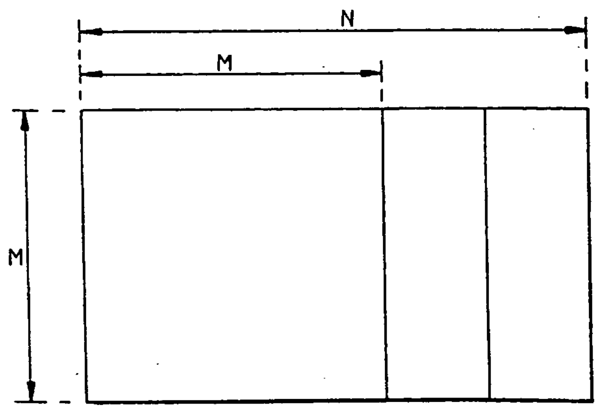
【解決手段】 各情報列がN－M個の有用な情報シンボルX_iとM個の冗長情報シンボルY_mとに分解されるN個のシンボルからなり、各コードが、N列からなり、各列内にt個の非ゼロシンボルを有するM＝N－K行からなる検査マトリクスAにより定義され、

－ 検査マトリクスAの全ての行に、同じ数の非ゼロシンボルを割り当て、

－ シンボルの数tを可能な限り小さい奇数にとり、

－ 検査マトリクスAの任意の2列が、多くても1つのみの非ゼロ値を有するような方法で、列を定義し、

－ 検査マトリクスAの2つの行が1つのみの非ゼロ共通値を有するような方法で、行を定義するプロセスを提案する。



【特許請求の範囲】
【請求項１】　バイナリ情報列を保護するためのＬＤＰＣコードを構築するプロセスであって、前記各情報列がＮ－Ｍ個の有用な情報シンボルX_iとＭ個の冗長情報シンボルY_mとに分解されるＮ個のシンボルからなり、各コードが、Ｎ列からなり、各列内にｔ個の非ゼロシンボルを有するＭ＝Ｎ－Ｋ行からなる検査マトリクスＡにより定義され、
－　検査マトリクスＡの全ての行に、同じ数の非ゼロシンボルを割り当て、
－　シンボルの数ｔを可能な限り小さい奇数にとり、
－　検査マトリクスＡの任意の２列が、多くても１つのみの非ゼロ値を有するような方法で、列を定義し、
－　検査マトリクスＡの２つの行が１つのみの非ゼロ共通値を有するような方法で、行を定義することを特徴とするプロセス。
【請求項２】　－　Ｐ行Ｐ列のｎ個のサブマトリクスを形成するために、Ｍ行Ｎ列の検査マトリクスＡを、Ｍ行Ｐ列のｎ個のサブマトリクスに再分割し、m^2個の左側サブマトリクスをＭ×Ｍサブマトリクスに、他の部分をｎ－ｍ個のＭ×Ｐサブマトリクスにグループ化し、
－　自己相関と列ベクトルｗの周期的な相互相関とを行うことにより、ｔ個の非ゼロ値と、（Ｍ－ｔ）個のゼロ値とを具備する長さＭの列ベクトルｗ［０．．．ｎ－１］のＭ×ｎ個の数列を定義することを特徴とする請求項１記載のプロセス。
【請求項３】　－　Ｐ行Ｐ列のｎ×ｍ個のサブマトリクスを形成するために、Ｍ行Ｎ列の検査マトリクスＡをＭ行Ｐ列のｎ個のサブマトリクスに再分割し、m^2個の左側サブマトリクスをＭ×Ｍ個のサブマトリクスに、他の部分をｎ－ｍ個のＭ×Ｐサブマトリクスにグループ化し、
－　ｔ個の非ゼロ値と（Ｍ－ｔ）個のゼロ値とを有する長さＭの列ベクトルｗ［０．．．ｎ－ｍ］のｎ－ｍ＋１個の数列を定義し、
－　シフトされた数列ｗ［０］が０または１点における場合を除いてシフトされていない数列自体と一致しないような方法で、第１の数列ｗ［０］が０、１、または、値ｔに等しい周期的な自己相関によって得られ、
－　ｎ－ｍ個の以下のシーケンスｗ［ｉ］［ｋ］が、
－　ｍの倍数によってシフトされた数列ｗ［ｉ］の値が、そのシフトされていない数列自体と決して一致しないような方法で、ゼロ値または値ｔに等しい周期的な自己相関によって、
－　かつ、ｍの倍数によってシフトされまたはシフトされていない数列ｉが、０または１点における場合を除いて数列ｊと一致しないような方法で、ｍによって多数回シフトするための数列ｗ［１．．．ｎ－ｍ］を用いて、ゼロまたは１の値の周期的な相互相関によって得られることを特徴とする請求項１記載のプロセス。
【請求項４】　有用な情報X_iのコーディングのために、検査マトリクスＡに、Ｎ－Ｍ個の情報シンボルX_iを表す列ベクトルをかけた積に等しいスイッチングマトリクスZ_mを決定し、該情報シンボルに、スイッチングマトリクスZ_mと検査マトリクスの寸法Ｍ×Ｍの部分の反転に等しいマトリクスＢとをかけた積の結果として得られる冗長シンボルY_mを追加することを特徴とする請求項１から請求項４のいずれかに記載のプロセス。

【発明の詳細な説明】
【０００１】
【発明の属する技術分野】この発明は、「低密度パリティ検査」を表す略語ＬＤＰＣとして知られるコードの構築およびコーディングのための、簡易かつシステマティックなプロセスに関するものである。
【０００２】
【従来の技術】１９６３年頃に提案されたギャラガーのコードは、現在、ターボコード(turbocode)に代わるものと考えられるＬＤＰＣコードの元祖である。M.J.C. MacKayによる"Good Error Correcting Codes Based on very Sparse Matrices"と題された、IEEE journal Transaction on Information Theory, Vol.45 No.2, March 1999に公開された記事は、これらのコードに関して興味深い結論、特に、
－　大きなサイズのブロックに対して、それらは、漸近的に「非常に良好なコード」であり、
－　重み付デコーディング（または、「ソフトデコーディング」あるいは「フレキシブルデコーディング」）を実行することが容易であるという事実を示している。しかしながら、それらを構築するためには帰納的な方法以外には方法は存在しない。
【０００３】このコーディング技術によれば、コード（Ｎ，Ｋ）は、Ｋ個の自由なシンボルを含むＮ個のシンボルが、Ｍ＝Ｎ－Ｋ行、Ｎ列からなる、そのパリティ検査マトリクスＡによって定義される。
【０００４】この検査マトリクスＡは、その低い「密度」を特徴とし、このことは、少数の非ゼロ要素を有することを意味する。さらに詳細には、このマトリクスＡは、ちょうどｔ個の非ゼロシンボルを各列内に有し、他の全ての要素はゼロである。
【０００５】コードワードのシンボルが、c_i，$i=0 \cdots N-1$と表示され、検査マトリクスの要素がA_{ij}と表示されるならば、コードは、
【数１】
$$\sum_{i=0....N-1} A_{mi} c_i \quad ここで\ m=0..M-1$$
の形態のＭ＝Ｎ－Ｋ個の関係を満足する。
【０００６】
【発明が解決しようとする課題】M.J.C. MacKayにより提案された方法は、より小さい単位マトリクスまたは三重対角マトリクスから初期マトリクスＡを構築し、その

後、所望の結果に到達するように、それらの列を入れ替えるものである。しかしながら、経験的には、それらの構造に対して印加される種々の拘束条件を満足することは困難であることが示されている。この発明の目的は、上述した欠点を緩和することである。

【０００７】

【課題を解決するための手段】この目的を達成するために、この発明は、Ｋ個の自由なシンボルを含むN個のシンボルを具備するＬＤＰＣコードを構築するためのプロセスであって、各コードをM＝N－K行、N列の、各行にｔ個の非ゼロシンボルを有する検査マトリクスＡにより決定し、

ａ　－　検査マトリクスＡの全ての列に、同じ数の非ゼロシンボル「ｔ」を割り当て、

ｂ　－　シンボル「ｔ」の数として、可能な限り小さい奇数を採用し、

ｃ　－　検査マトリクスＡの任意の２つの列が、多くても１つの非ゼロ値を有するように、各列を定義し、

ｄ　－　検査マトリクスの２つの行が１つのみの非ゼロ共通値を有するように、各行を定義することを特徴としている。

【０００８】この発明に係るプロセスは、可能な限り低密度を有すると同時に、その必要な計算電力が数ｔに比例する妥当な複雑さに対して、良好な性能を与える検査マトリクスＡを使用することにより、コーディングおよびデコーディングアルゴリズムを簡易化することができるという利点を有する。いくらかの誤差が存在するという限りにおいて、上記拘束条件”ｃ”は、全ての情況において、デコーディングアルゴリズムを収束させることができる。

【０００９】

【発明の実施の形態】この発明の他の特徴および利点は、添付図面に関連して以下に示された説明により明らかになる。図１は、検査マトリクスＡの分割を示す配列である。図２および図３は、図１の配列のm^2個の左側のサブマトリクスの、M×Mサブマトリクス、および、ｎ－m個のM×Ｐサブマトリクスへのグループ化を示し＊ている。図４および図５は、この発明に係るプロセスの第１および第２の変形例に従って、それぞれ得られた検査マトリクスＡを示している。図６～図９は冗長シンボルを計算するスイッチングマトリクスを示している。

【００１０】この発明に係るプロセスを実施するために、検査マトリクスＡが、図１に示されるように、N＝ｎＰ，M＝ｍＰであり、ｎおよびmが、互いに素の数となるように、ｎ個のM行Ｐ列のサブマトリクス、または、ｍ×ｎ個のＰ行Ｐ列の正方サブマトリクスに再分割される。

【００１１】m^2個の左側サブマトリクスは、その後、図２に示されるように、M×Mサブマトリクス（このサブマトリクスは、コーディングアルゴリズムを非常に簡単にすることができる。）にまとめられ、他のサブマトリクスはｎ－m個のM×Ｐサブマトリクスにグループ化される。この構築プロセスは、ｍ＝１またはｍ＝ｔによる２つの変形例に従って、以下に説明される。

【００１２】異なる値のｍは、ｔｒが整数であることを要求する条件”ａ”のために、ここでは考慮しない。すなわち、ｔｒ＝ｔN／Mまたはｔｒ＝ｔｎ／ｍである。ｎおよびｍは、互いに素であり、ｔはｍで割り切れなければならず、したがって、ｍは、素数でありかつ小さいｔに対しては、１またはｔに等しい場合のみが可能である（小さい値のｔ、すなわち、３，５，７に対して真。）。

【００１３】ｍ＝１（冗長性ｒ／（ｒ－１）を有するコード）の第１の変形例において、この発明に係るプロセスは、冗長シンボルの数が正確にN／ｒ（ｒは整数）個であるｒ／（ｒ－１）の形態の冗長度N／Kを有するコードに対して当てはまる。この場合には、MはＰに等しく、図２の配列は、図３の配列に減じられる。したがって、プロセスは、ｔ個の１と（M－ｔ）個の０とからなる長さMのｎ個の数列を検索することになる。

【００１４】したがって、以下、ｗ［０．．ｎ－１］と表されるこれらの数列は、

－　全てのｉ＝０．．．ｎ－１に対して、

【数２】

$$\sum_{k=0 \ldots M-1} w[i][k]\ \ w[i][k] = t \quad \text{（定義により）}$$

$$\sum_{K=0 \ldots M-1} w[i][k]\ \ w[i][(k+p) \text{ modulo } M] = 0 \text{ 又は } 1, \text{ ここで } p = 1..M-1$$

となるような、０，１またはｔに等しい周期的な自己相関（シフトされた数列ｉは、０または１点における場合を除き、シフトされていないその数列ｉ自体と一致しない）、※

※－　ｉ＝０．．ｎ－１，ｊ＝０．．ｎ－１が異なる全ての対｛ｉ，ｊ｝に対して、

【数３】

$$\sum_{K=0 \ldots M-1} w[i][k]\ \ w[j][(k+p) \text{ modulo } M] = 0 \text{ 又は } 1, \text{ ここで } p = 0..M-1$$

となるような、０または１に等しい周期的な相互相関（シフトされた数列ｉは、０または１点における場合を除き、数列ｊと一致しない）により得られる。

【００１５】数列ｗを計算するアルゴリズムは非常に簡易である。このアルゴリズムはｐｏｓ［ｘ］［０］＝０，ｐｏｓ［ｘ］［１］＝１，．．．，ｐｏｓ［ｘ］［ｔ－１］＝ｔ－１から始まって、自己相関および相互相関条件を満たすように、それらを修正することによ

り、これらの数列が「１」を有する位置ｐｏｓ［０］［０．．．ｔ−１］，ｐｏｓ［１］［０．．．ｔ−１］，．．．，ｐｏｓ［ｎ−１］［０．．．ｔ−１］を連続して決定する。ｔ＝３に対して、使用される計算ループは、付録１に示されている。

【００１６】このアルゴリズムは、ｎが大きすぎると失敗する。所定のＭに対して、いくつかの「小さい」コードが発見されるが、一般には大きなサイズ（Ｎ＞＞１００）のコードが求められるので、このことはあまり重要ではない。

【００１７】したがって、マトリクスＡの列は、非常に簡単に、周期的に入れ替えられたベクトルｗであり、

−　ｋ番目のサブマトリクス（ｋ＝０．．．ｎ−１）は、

Ａ［行］［列］＝ｗ［ｋ］［（行−（列−ｋＰ））modulo　Ｍ］

ここで、行＝０．．．Ｍ−１、列＝ｋＰ．．．（ｋ＋１）Ｐ−１である。

【００１８】したがって、マトリクスＡの各行は、正確にｔ個の非ゼロ値を、ｎ個のサブマトリクスの各々に有し、すなわち、ｎの総数＝ｔｒである。

【００１９】ＬＤＰＣコード（７５，５０，ｔ＝３，ｔｒ＝９）、冗長度３／２（ｒ＝ｎ＝３）、Ｐ＝２５に対＊＊して、このプロセスにより得られる、例としてのマトリクスＡが、図４に示されている。ここに示された配列によれば、

ｗ［０］［ｉ］＝１　ここで　ｉ＝０，１，３

ｗ［１］［ｉ］＝１　ここで　ｉ＝０，４，．９

ｗ［２］［ｉ］＝１　ここで　ｉ＝０，６，１３

である。

【００２０】提案された構造は、

−　各列が、正確にｔ個の非ゼロ値（ベクトルｗの定義による）を有し、

−　各行が、正確にｔｒ個の非ゼロ値（ベクトルｗの自己相関および相互相関特性による）を有し、

−　全ての別々の列の対が、多くても１つの共通の非ゼロ値（同上）を有し、

−　全ての別々の行の対が、多くても１つの共通の非ゼロ値（同上）を有することを保証する。

【００２１】ｍ＝ｔの場合に対応する上記変形例により示唆された第２の変形例によれば、この発明に係るプロセスは、ｗ［０．．ｎ−ｍ］と表される、ｔ個の「１」と（Ｍ−ｔ）個の「０」とからなる長さＭのｎ−ｍ＋１個の数列を検索する。

【００２２】第１の数列ｗ［０］は、

【数４】

$$\sum_{K=0 \ldots M-1} w[0][k]w[0][k] = t \quad \text{（定義により）}$$

$$\sum_{K=0 \ldots M-1} w[0][k]w[0][(k+p)\ \mathrm{modulo}\ M] = 0 \quad \text{又は} \quad 1,$$

$$\text{ここで、}\ p=1..M-1$$

のように、（シフトされた数列０が、０または１点における場合を除き、そのシフトされていない数列自体と一致しない）０，１またはｔに等しい周期的な自己相関によって得られる。※

※【００２３】実際には、ｍ＝１に対するものと同じ定義である。以下の数列ｗ［１．．．ｎ−ｍ］は、

−　全てのｉ＝１．．．ｎ−ｍに対して、

【数５】

$$\sum_{K=0 \ldots M-1} w[i][k]w[i][k] = t \quad \text{（定義により）}$$

$$\sum_{K=0 \ldots M-1} w[i][k]w[i][(k+pm)\ \mathrm{modulo}\ M] = 0, \text{ここで}\ p=1..P-1$$

のように、ｍの倍数のシフト（ｍの倍数によりシフトされた数列ｉは、シフトされていないもの自体と決して一致しない。）に対して０またはｔに等しい周期的な自己★

★相関、

−　全てのｉ＝１．．．ｎ−ｍに対して、

【数６】

$$\sum_{K=0 \ldots M-1} w[i][k]w[0][(k+p)\ \mathrm{modulo}\ M]=0 \ \text{又は}\ 1, \text{ここで}\ p=0..M-1$$

のように、数列ｗ［０］（シフトされまたはシフトされていない数列ｉは、０または１点における場合を除き、数列０と一致しない。）を用いた、点０または１に等しい周期的な相互相関、☆

☆−　および、ｉ＝１．．．ｎ−ｍとｊ−１．．．ｎ−ｍとが異なる全ての対｛ｉ，ｊ｝に対して、

【数７】

$$\sum_{K=0 \ldots M-1} w[i][k]w[j][(k+pm)\ \mathrm{modulo}\ M]=0 \quad \text{又は} \quad 1, \ \text{ここで、}$$

$$p=0..P-1$$

のように、ｍの倍数のシフト（ｍの倍数によりシフトされまたはシフトされていない数列ｉは、点０または１における場合を除き、数列ｊと一致しない。）に対して、０または１に等しい数列ｗ［１．．ｎ−ｍ］を用いた周期的な相互相関によって得られる。

【００２４】数列ｗを計算するためのアルゴリズムは、

上述したものと同じである。自己相関および相互相関基準のみが変更され、Mの代わりに、P個の点においてのみ、それらを確かめることが必要である。

【００２５】したがって、マトリクスAの列は、１またはmに等しいステップを用いて、

－　AのM×M個の左側サブマトリクス：

A［行］［列］＝w［0］［（行－列）modulo M］

ここで、行＝０．．．M－１

列＝０．．．M－１

－　k＝m．．．n－１に対して、

A［行］［列］＝w［k－m＋１］［行－m（列－kP））modulo M］

ここで、行＝０．．．M－１

列＝kP．．．（k＋１）P－１

となるように、周期的に入れ替えられるベクトルwである。

【００２６】このように、マトリクスAの各行は、その最初のM個の列に、正確にm＝t個の非ゼロ値、したがって、P個の連続した列のn－m個のパケット毎に１個の非ゼロ値を有し、すなわち、nの総数またはtrである。＊

＊【００２７】この発明に係るプロセスの第２の変形例により得られた例示したマトリクスAは、LDPCコード（７５，３０，t＝３，tr＝５）、冗長度５／２（n＝５，m＝３）、P＝１５に対して、図５に示される。図５の配列において、

w［0］［i］＝1　ここで　i＝0，1，3

w［1］［i］＝1　ここで　i＝0，4，8

w［2］［i］＝1　ここで　i＝0，5，10

であることを明記しておく。

【００２８】２つの変形例に基づいて示されたこの発明に係るプロセスは、冗長シンボルY_iおよび情報シンボルX_iをコーディングするための非常に簡易なアルゴリズムに直接つながるものである。このためには、冗長シンボルY_iをコードワードの最初のM個のシンボルであると考え、自由シンボルX_i（情報）を最後のN－M個のシンボルであると考えるだけで十分である。

【００２９】したがって、全てのコードワードによって満足されなければならない等式は、以下の形態に書き直すことができる。

【数８】

$$\sum_{i=0...M-1} Ami\ Yi + \sum_{i=M...N-1} Ami\ Xi = 0, \text{ここで } m=0...M-1$$

又は、

$$\sum_{i=0...M-1} Ami\ Yi = Zm, \text{ここで } m=0...M-1$$

又、ここで、

$$Zm = -\sum_{i=M...N-1} Ami\ Xi, \text{ここで } m=0...M-1$$

【００３０】したがって、プロセスは、最初に、スイッチングマトリクスのM個の量Z_mを計算し、その後に冗長シンボル：

【数９】

$$Ym = \sum_{i=0...M-1} Bmi\ Zi, \text{ここで } m=0...M-1$$

を計算する。

【００３１】例えば、（７５，５０）というLDPCコードに対して、量Z_mは、図６の配列により定義された等式システムを用いて計算され、該図６の配列は、その解法の後に、図７の冗長シンボルの配列に変換される。

【００３２】一般的な要素B_{ij}を有するマトリクスBは、マトリクスAの（寸法M×Mの）左側部分A_Mを反※

※転したものである。該マトリクスAは、構築により、非常に簡易な形態を有し、その全ての列は、数列w［0］［０．．M－１］の周期的な繰り返しである。

A_{ij}＝w［0］［（i－j）modulo M］，i＝０．．M－１，j＝０．．M－１

【００３３】したがって、マトリクスBは、単一の行b［０．．M－１］の周期的な入れ替えであるM個の行から構成されている。すなわち、

B_{ij}＝b［（j－i）modulo M］

Bは、A_Mの反転であり、係数bは以下のように定義される。

【数１０】

$$\sum_{i=0...M-1} w[0][i]b[(i+k)\ \text{modulo}\ M] = 1\ (k=0\text{の場合})$$
$$,\ 0\ (k=1...M-1\text{の場合})$$

【００３４】例えば、（７５，５０）というLDPCコードに対して、冗長係数Y_mは、解法後に図９の配列に変換される図８の配列により定義される式のシステムを介して計算される。

【００３５】しかしながら、計算が不可能な場合がある。以下の形態で定義される等式を記述することは、実際には不可能である。

$^{t}A_{M}t$ {b［０］，b［１］，．．．，b［M－1］}＝t｛1，0，0，．．．，0｝

【００３６】マトリクス$^{t}A_{M}$は、循環マトリクスであり、その第１行は、a［０．．．M－１］＝w［０］に等しい。その行列式は、そのM個の固有値｜０．．．M－１の積に等しい。

【００３７】ｋ番目の固有値は、それ自体、以下の式で与えられる。

【数１１】

$$\lambda_k = \sum_{i=0 \ldots M-1} a[i]\alpha_{ik}$$

ここで、αは、１のM乗根である。

【００３８】例えば、

－　w［０］＝a［０．．．M－１］＝｛１，１，０，１，０，０，０，．．．．｝に対し、

－　バイナリコード（これらは、加算が排他的OR（XOR）に等しく、積算が論理ANDに等しいガロア体CG（２）に存在する）に対して、$\lambda_1 = 1 + \alpha + \alpha^3$を得る。

【００３９】Mが７の倍数である場合には、等式$1 + \alpha + \alpha^3 = 0$は、αが１の７乗根であるガロア体（多項式$g(x) = 1 + x + x^3$は単純化できず、かつ、CG（２）において原始的なものであり、ガロア体CG（２３）を生じる。）を定義し、$\lambda_1 = 0$であることを意味している。

【００４０】したがって、提案されたアルゴリズムにより発見されたＬＤＰＣコードの内、

－　$^{t}A_{M}$の固有値の１つがゼロとなり、

－　したがって、その行列式がゼロとなり、

－　したがって、適当な値b［ｉ］を発見することができず、

－　したがって、（Y_{is}を計算するための）コーディングを行うことができないので、このw［０］を保持する場合には、Mが７の倍数であるものを消去する必要がある。

【００４１】ごく一般的には、w［０］に対してなされる選択にかかわらず、コーディングが実行されることを許容しないために、適当ではないMの値が存在することになる。（$x^M - 1$および

【数１２】

$$a(x) = \sum_{i \ldots M-1} a[i] \quad x^k$$

を因数分解することにより、）これらのMの値が、a（x）で$x^{M0} - 1$が割り切れる値M０の倍数であることが容易に示される。

【００４２】例えば、ｔ＝３によるバイナリコードに対して、

－w［０］＝｛１，１，０，１，．．．｝

－w［０］＝｛１，０，１，１，．．．｝

は、Mが７の倍数であることを禁止する（a（x）は１の７乗根(7th root of unity)を定義する）。

－w［０］＝｛１，１，０，０，１，．．．.｝

－w［０］＝｛１，０，０，１，１，．．．｝

は、Mが１５の倍数であることを禁止する（a（x）は１の１５乗根を定義する）。

－w［０］＝｛１，０，１，０，１，．．．｝は受け入れられない（正しくない自己相関）。

－w［０］＝｛１，１，０，０，０，１，．．．｝

－w［０］＝｛１，０，０，０，１，１，．．．｝

は、Mが３の倍数であることを禁止する（a（x）は$1 + x + x^2$の倍数であり、１の立方根を定義する）。

－w［０］＝｛１，０，１，０，０，１，．．．｝

－w［０］＝｛１，０，０，１，０，１，．．．｝

は、Mが３１の倍数であることを禁止する（a（x）は１の３１乗根を定義する）。

【００４３】係数b［ｉ］は、以下のようにして計算される。Mの禁止されていない値に対して、a［ｉ］（またはw［０］［０．．．M－１］）からb［ｉ］を計算するための特に簡易なアルゴリズムが存在する。このアルゴリズムは、係数a［M－１，M－２，．．．，１，０］を有する有限時間インパルス応答フィルタ(FIR)A（z）により、時間分割されかつフィルタリングされた後に、一連のb［ｉ］が時分割数列｛１，０，０，．．．｝を与えなければならないという観測によるものである。実際には、前に枚挙されたw［０］の内の１つを使用したバイナリコードに対して、この数列が、長さ７（または１５または３１または６３）の最大数列（最大長数列）の連鎖によって形成される。

【００４４】したがって、無限インパルス応答フィルタ(IIR)１／A（z）のインパルス応答が計算され、一旦時分割されると、A（z）によるフィルタリング後に数列｛１，０，０，．．．｝を与える長さMのスライスがそこから抽出される。

【００４５】例えば、ｔ＝３のバイナリコードに対して、かつ、a［０］，a［ｋ１］およびa［ｋ２］のみがゼロではないものに対して、相当するアルゴリズムが、付録２に提供されている。

【００４６】各コーディング時に前の計算を行わないように簡易化する目的から、コーディングアルゴリズムは、循環（recurrence：１／A（z）によるフィルタリング）によって、全ての他のものを再計算可能とする、最後のｋ２個の要素b［M－ｋ２．．．M－１］により定義されさえすることができる。

【００４７】標準コーディングの第２段階と同様に、アルゴリズム（ZからのYの計算）は、平均して、大きなコードに対して相当な回数となるM^2／２回の操作を具備し、その複雑さは、サイズの二次関数となり、さら

に、中間配列Ｚ（Ｍ個の要素）を格納し、配列ｂ（これもＭ個の要素）を知ることが必要であるので、その場で計算されない場合には、アルゴリズムのこの部分は、（例えば、（７５，５０）ＬＤＰＣコードに対する）図９の配列により示された方法で、Ｙを与える等式を再記述することによって、非常に小さいサイズの２つの中間配列のみを使用するように修正されてもよい。

【００４８】（ｔ＝３に対して）最初のＭ－ｋ２行は、解法前にＹを与える等式のシステムの最後のＭ－ｋ２行である。最後のｋ２行は、解法後にＹを与える等式のシステムの最後のｋ２行である。したがって、Ｙを逆順に、すなわち、Ｙ［Ｍ－１］，Ｙ［Ｍ－２］，．．．，Ｙ［０］の順に計算するだけで十分である。

【００４９】したがって、行われるべき操作の回数は、平均して、ｋ２Ｍ／２回（Ｙ［Ｍ－１］．．．Ｙ［Ｍ－ｋ２］の計算）とその後のｔ（Ｍ－ｋ２）回（他の全ての計算）、すなわち、約（ｔ＋ｋ２／２）Ｍ回であり、これにより、その複雑さは、サイズの線形関数のみとなる。アルゴリズムは、入力としてＸ［Ｍ．．．ｎ］を使用する。

【００５０】Ｘの下部（Ｘ［０．．．Ｍ－１］）は、Ｚのための一時的な格納場所として使用され、Ｘ［０．．．Ｍ－１］は、最終段階における循環シフトを回避するように、Ｚ［ｋ２．．．Ｍ－１，０．．．ｋ２－１］を格納する。ｂ［ｉ］は、その場で、ｂ［Ｍ－ｋ２．．．Ｍ－１］から繰り返して計算される。

【００５１】コードは、以下の２つの配列により定義される。

－　ｂの最後のｋ２個の要素からなる配列ｅｎｄＢ［０．．．ｋ２－１］と、

－　数列ｗ［０］，ｗ［１］，．．．，ｗ［ｎ－ｍ］の非ゼロ要素の位置を含む配列ｐｏｓ［０．．．（ｎ－ｍ＋１）ｔ］である。

【００５２】以下の２つのサイズｋ２からなる内部バッファが使用される。

－　ｂ［ｉ］を計算するためのｒｅｇ［０．．．ｋ２－１］と、

－　Ｙ［Ｍ－ｋ２．．．Ｍ－１］の中間値を格納するためのｔｅｍｐ［０．．．ｋ２－１］である。したがって、高速コーディングのための完全なアルゴリズムが、付録３に示されている。

【００５３】これらのアルゴリズムは、非常に簡易に実行することができる。特に、非常に少ないパラメータ、すなわち、数列ｗ内の「１」の（ｎ－ｍ＋１）（ｔ－１）個の非ゼロ位置、および、あるいはｋ２個のコーディング係数によって、コードを定義する特徴を有している。それらのアルゴリズムは、条件ａ－ｄ（例えば、長さＰ＝２５のｎ＝６個の数列ｗを必要とする、冗長度６／５の（１５０，１２５）コードではないこと）に合致する可能なコードの全てを与えるものではないが、ＮおよびＫが事前に定義されている任意のアプリケーションにおいて、

－　ＮＬＤＰＣ＝Ｎ、ＫＬＤＰＣ＝Ｋを有する（ＮＬＤＰＣ，ＫＬＤＰＣ）コード、または、

－　任意にゼロに設定されたｄ個の有用なシンボルの非伝達により短縮される微小のｄを有する（ＮＬＤＰＣ＋ｄ，ＫＬＤＰＣ＋ｄ）近接コードのいずれかを発見することができる。

【００５４】例えば、冗長度５／３（率０．６）のコード（Ｎ，Ｋ）を得るためには、ｄ＝ＮＬＤＰＣ／１５を有する冗長度８／５（率０．６２５）の（ＮＬＤＰＣ＋ｄ，ＫＬＤＰＣ＋ｄ）コードから開始するだけで十分である。５００以下のＮ値およびｔ＝３に対して、以下の冗長度を有する９３２個の異なるコードを極めて迅速に構築することが可能である（ここでは、故意に、４～８／７の間に配される冗長度およびｋ２＝３に対してｗ［０］＝｛１，１，０，１，０，０，０，．．．｝のコードに制限した。）。

Ｒ＝４／１	すなわち、	４．０００	（１０５コード）
Ｒ＝５／２	すなわち、	２．５００	（８２　コード）
Ｒ＝６／３または２／１	すなわち、	２．０００	（２０３コード）
Ｒ＝７／４	すなわち、	１．７５０	（５５　コード）
Ｒ＝８／５	すなわち、	１．６００	（４７　コード）
Ｒ＝９／６または３／２	すなわち、	１．５００	（１２４コード）
Ｒ＝１０／７	すなわち、	１．４２８	（３４　コード）
Ｒ＝１１／８	すなわち、	１．３７５	（２８　コード）
Ｒ＝１２／９または４／３	すなわち、	１．３３３	（８４　コード）
Ｒ＝１３／１０	すなわち、	１．３００	（２０　コード）
Ｒ＝１４／１１	すなわち、	１．２７３	（１７　コード）
Ｒ＝１５／１２または５／４	すなわち、	１．２５０	（５６　コード）
Ｒ＝１６／１３	すなわち、	１．２３１	（１１　コード）
Ｒ＝１７／１４	すなわち、	１．２１４	（７　コード）
Ｒ＝１８／１５または６／５	すなわち、	１．２００	（３４　コード）
Ｒ＝１９／１６	すなわち、	１．１８７	（３　コード）

13 14

R＝２０／１７　すなわち、１．１７６　（２　コード）
R＝２１／１８または7／6　すなわち、１．１６７　（１７　コード）
R＝２４／２１または8／7　すなわち、１．１４３　（３　コード）

【００５５】さらに、５００以下のNの所定値に対して、（N＝４８０に対して）１２個までの異なるコードが存在する。例えば、Nが２８８以上の６の倍数になるとすぐに、長さN、冗長度６／５，３／２，２／１の３つのコード、例えば、LDPC（３００，２５０）＋LDPC（３００，２００）＋LDPC（３００，１５０）が常に存在する。

【００５６】このことは、それぞれが長さNを有し、異なる感度を有する３つのバイナリ数列から構成されたバイナリ数列を効果的に保護するのに非常に有用である。もちろん、例えば、マトリクスAの行および／または列のランダム順列のような、これらのアルゴリズムの種々の変形例を想定することも常に可能である。また、非バイナリコードへの適合が特に簡易であることも重要である。

【００５７】

【付録】

付録１

```
        for(x=0; x<n - m + 1; x++){
        pos[x][0] = 0;
        for(pos[x][1]=pos[x][0]+1; pos[x][1]<M-1;
        pos[x][1]++){
            for(pos[x][2]=pos[x][1]+1; pos[x][2]<M;
            pos[x][2]++){
              (if the conditions are not satisfied,
                     continue
                     otherwise, go to ok)
            }
        }
        (stop: impossible to find a suitable choice
        for pos[x][0...t-1])
ok:;
        }
```

付録2

(language C: the operator "^" corresponds to EXCLUSIVE OR.

```
/* Initialization of the b pass, of length M*/
for(i-M-k2; i<M; i++)
   b[i]=0;

/* Calculation of N successive values of the
impulse response of 1/A(z)*/
b[0] = 1;
for(i=1; i<k2; i++)
   b[i] = b[(i+M-(k2-k1)) % M]^b[(i+M-k2) % M];
for(i=k2; i<M; i++)
   b[i] = b[i-(k2-k1)]^b[i-k2];

/* Arrange for there to be just one 1
in the last k2 positions of b filtered by A(z)*/

weight = 0; /* all except 1 */

while(weight != 1){

  /* Shift by one notch */
  for(i=1; i<M; i++)
    b[i-1] = b[i];
  b[M-1] = b[M-1-(k2-k1)]^b[M-1-k2];
  /* Verify */
  weight = 0;
  for(i=M-k2; i<M; i++){
    char sum = b[i]^b[(i+k1)%M]^b[(i+k2)%M];
    if(sum){
            shift = M - i;
            weight++;
    }
  }

  /* Particular case where M is forbidden */
  if(weight ==0)
            return(M_FORBIDDEN);
```

```
}

/* rightward final circular shift:
b[i]=b[(i - shift) % M]*/
for(dec=0; dec<shift; dec++){
      char temp = b[M-1];
      for(i=M-1; i>0; i--)
               b[i]=b[i - 1];
      b[0] = temp;
}

return(OK);
```

付録３

(language C):

```
/* Phase 1: calculation of M intermediate parities z.
These parities are calculated by reading the successive columns of the coding matrix, namely A[*][M],...,A[*][N]
They are placed at the head of x temporarily */

#define     z     x
    for(i=0; i<M; i++)
        z[i] = 0;

/* Loop over the n-m right submatrices of A*/
c0 = M;
c1 = c0 + P;
```

```
for(k = 1; k<=n - m; k++){
    offset = 0;
    for(c=c0; c<c1; c++){
        if(x[c]!=0)
            for(i=0; i<t; i++){
              /*    p    ought    to    be
              offset + pos[i].
              We decrement it by k2 to avoid
shifting

              the array z before phase 3*/
              p = offset + pos[k*t+i] - k2;
              if(p<0)
                   z[p + M] = z[p + M]^1;
              else
                   if(p<M)
                        z[p] = z[p]^1;
                   else
                        z[p - M]=z[p - M]^1;
              }
        offset = offset + m;
}
```

```
        c0 = c1;
        c1 = c1 + P;
    }

    /* Phase 2: calculation of the last k2 parity
    symbols */
    ixb0 = M - 1 - k2;

    /*1: initialization of the last k2 elements of y
    temp[0...k2-1]=y[M-1, M-2, ...M-k2]*/
    for(k=0; k<k2; k++)
       temp[k] = 0;

    /*2: copy over the last k2 elements of b
    reg[0...k2-1] = b[M-k2...M-1]*/
    for(i=0; i<k2; i++)
       reg[i] = finB[i];

    /*3: iterative calculation of the last k2 symbols
    */
    for(i=0; i<M; i++){
       /* b[i] = {1 0 0 ...}^b[i-(k2-k1)]^b[i-k2]
       with b[i-k2]...b[i-1] = reg[0...t2-1]

We must verify that:
b[-k2] + b[k1-k2] + b[0] = 0
...
b[-2] + b[k1-2] + b[k2-2] = 0
b[-1] + b[k1-1] + b[k2-1] = 0
b[0] + b[k1] + b[k2] = 1
b[1] + b[1+k1] + b[1+k2] = 0
... */
if(i==k2)
    input = 1;
else
    input = 0;
bi = input^reg[0]^reg[k1];
for(k=1; k<k2; k++)
    reg[k - 1] = reg[k];
```

```
        reg[k2 - 1] = bi;

        if(bi != 0)
            for(k=0; k<k2; k++)
                    if(z[(ixb0 - k + M) % M]!=0)
                        temp[k] = temp[k]^1;
        ixb0 = ixb0 + 1;
        if(ixb0==M)
            ixb0 = 0;
    }

    /*4: The z values have already been left-shifted
    to avoid overwriting. Otherwise, it would be
    necessary to do:

    for(k=0; k<M - k2; k++)
       z[k] = z[k + k2];

    Copy over temp to the end of y */
#define y       x
    for(k=0; k<k2; k++)
       y[M - 1 - k] = temp[k];

    /* Phase 3: calculation of y[M-k2-1, M-k2-2,...,0]
    y[k+k2-k2] + y[k+k2-k1] + y[k+k2-0] + z[k+k2] = 0
    y[k] goes to x[k]
    z[k+k2] is in x[k]
    Hence:
    x[k+k2-k2] + x[k+k2-k1] + x[k+k2-0] + x[k] = 0
    i.e.:
    x[k+k2-k2] = -(x[k+k2-k1] + x[k+k2] + x[k])
    */
    for(k = M-k2-1; k>=0; k--)
       y[k] = y[k+k2-k1]^y[k+k2]^z[k];
```

【図面の簡単な説明】

【図１】　検査マトリクスAの分割を示す配列である。

【図２】　図１の配列のm^2左サブマトリクスのM×Mサブマトリクスへのグループ化を示す図である。

【図３】　図１の配列のm^2左サブマトリクスのM×Pサブマトリクスへのグループ化を示す図である。

【図４】　この発明に係るプロセスの第１の変形例に従う検査マトリクスAを示す図である。

【図５】　この発明に係るプロセスの第２の変形例に従う検査マトリクスAを示す図である。

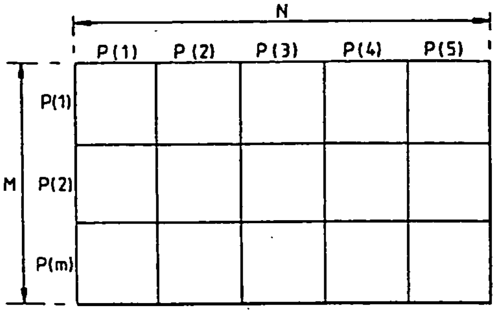
【図６】　冗長シンボルを計算するためのスイッチングマトリクスを示す図である。

【図７】　図６と同様のスイッチングマトリクスを示す図である。

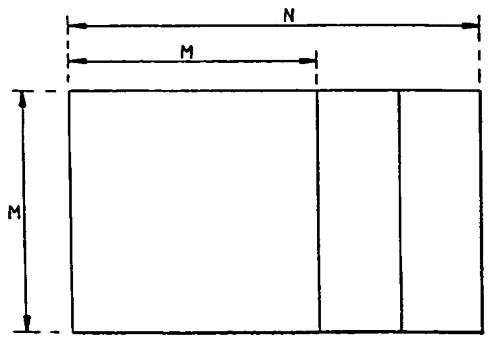
【図８】　図６と同様のスイッチングマトリクスを示す図である。

【図９】　図６と同様のスイッチングマトリクスを示す図である。

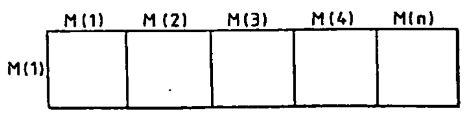
【図１】



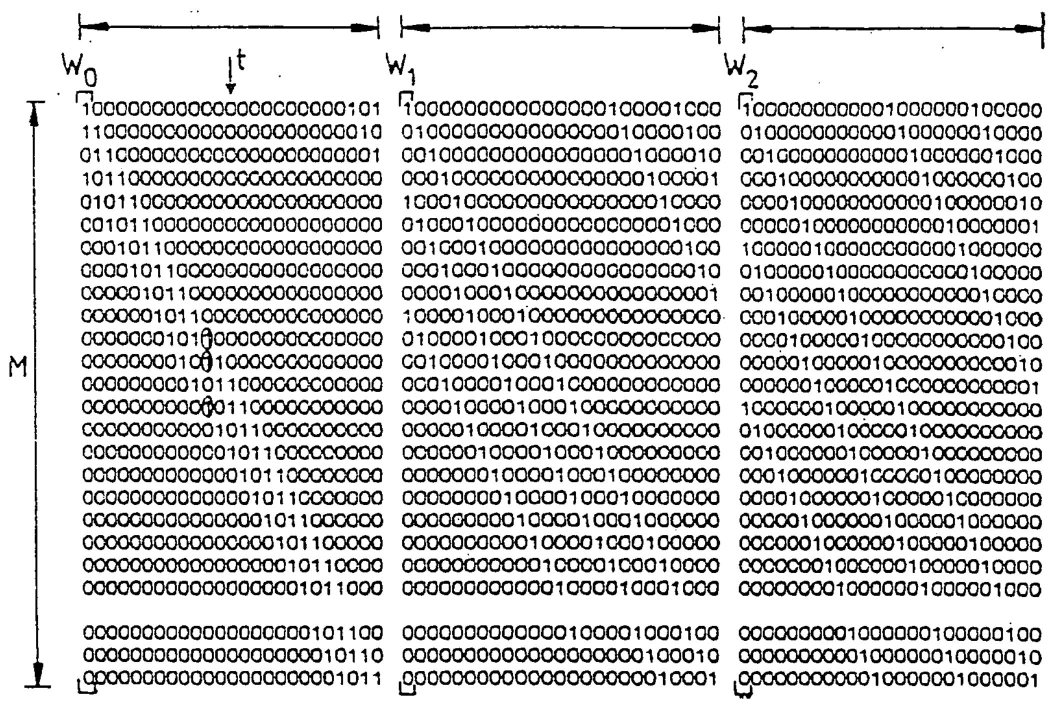
【図２】



【図３】



【図４】



【図５】

W_0 M	W_1 P	W_2 P
100000000000000000000000000000000000000000101	100000000000000	100000000000000
110000000000000000000000000000000000000000010	000000000000001	000000000000100
011000000000000000000000000000000000000000001	000000000000010	000000000000001
101100000000000000000000000000000000000000000	010000000000000	010000000000000
010110000000000000000000000000000000000000000	100000000000000	000000000000010
001011000000000000000000000000000000000000000	000000000000001	100000000000000
000101100000000000000000000000000000000000000	001000000000000	001000000000000
000010110000000000000000000000000000000000000	010000000000000	000000000000001
000001011000000000000000000000000000000000000	100000000000000	010000000000000
000000101100000000000000000000000000000000000	000100000000000	000100000000000
000000010110000000000000000000000000000000000	001000000000000	100000000000000
000000001011000000000000000000000000000000000	010000000000000	001000000000000
000000000101100000000000000000000000000000000	000010000000000	000010000000000
000000000010110000000000000000000000000000000	000100000000000	010000000000000
000000000001011000000000000000000000000000000	001000000000000	000100000000000
000000000000101100000000000000000000000000000	000001000000000	000001000000000
000000000000010110000000000000000000000000000	000010000000000	001000000000000
000000000000001011000000000000000000000000000	000100000000000	000010000000000
000000000000000101100000000000000000000000000	000000100000000	000000100000000
000000000000000010110000000000000000000000000	000001000000000	000100000000000
000000000000000001011000000000000000000000000	000010000000000	000001000000000
000000000000000000101100000000000000000000000	000000010000000	000000010000000
000000000000000000010110000000000000000000000	000000100000000	000010000000000
000000000000000000001011000000000000000000000	000001000000000	000000100000000
000000000000000000000101100000000000000000000	000000001000000	000000001000000
⋮	⋮	⋮
000000000000000000000000000000000000000101100	000000000000001	000000000000001
000000000000000000000000000000000000000010110	000000000000010	000000000001000
000000000000000000000000000000000000000001011	000000000000100	000000000000010

【図６】

Z[0] Z[M-1]	X[M] X[M+P-1]	X[M+P] X[N]
100000000000000000000000	100000000000000100001000	100000000000100000100000
010000000000000000000000	010000000000000010000100	010000000000010000010000
001000000000000000000000	001000000000000001000010	001000000000001000001000
000100000000000000000000	000100000000000000100001	000100000000000100000100
000010000000000000000000	100010000000000000010000	000010000000000010000010
000001000000000000000000	010001000000000000001000	000001000000000001000001
000000100000000000000000	001000100000000000000100	100000100000000000100000
000000010000000000000000	000100010000000000000010	010000010000000000010000
000000001000000000000000	000010001000000000000001	001000001000000000001000
000000000100000000000000	100001000100000000000000	000100000100000000000100
000000000010000000000000	010000100010000000000000	000010000010000000000010
000000000001000000000000	001000010001000000000000	000001000001000000000001
000000000000100000000000	000100001000100000000000	100000100000100000000000
000000000000010000000000	000010000100010000000000	010000010000010000000000
000000000000001000000000	000001000010001000000000	001000001000001000000000
000000000000000100000000	000000100001000100000000	000100000100000100000000
000000000000000010000000	000000010000100010000000	000010000010000010000000
000000000000000001000000	000000001000010001000000	000001000001000001000000
000000000000000000100000	000000000100001000100000	000000100000100000100000
000000000000000000010000	000000000010000100010000	000000010000010000010000
000000000000000000001000	000000000001000010001000	000000001000001000001000
000000000000000000000100	000000000000100001000100	000000000100000100000100
000000000000000000000010	000000000000010000100010	000000000010000010000010
000000000000000000000001	000000000000001000010001	000000000001000001000001

【図７】

```
Y[0]                 Y[M-1]   Z[0]                 Z[M-1]
100000000000000000000101   100000000000000000000000
110000000000000000000010   010000000000000000000000
011000000000000000000001   001000000000000000000000
101100000000000000000000   000100000000000000000000
010110000000000000000000   000010000000000000000000
001011000000000000000000   000001000000000000000000
000101100000000000000000   000000100000000000000000
000010110000000000000000   000000010000000000000000
000001011000000000000000   000000001000000000000000
000000101100000000000000   000000000100000000000000
000000010110000000000000   000000000010000000000000
000000001011000000000000   000000000001000000000000
000000000101100000000000   000000000000100000000000
000000000010110000000000   000000000000010000000000
000000000001011000000000   000000000000001000000000
000000000000101100000000   000000000000000100000000
000000000000010110000000   000000000000000010000000
000000000000001011000000   000000000000000001000000
000000000000000101100000   000000000000000000100000
000000000000000010110000   000000000000000000010000

000000000000000001011000   000000000000000000001000
000000000000000000101100   000000000000000000000100
000000000000000000010110   000000000000000000000010
000000000000000000001011   000000000000000000000001
```

【図８】

```
Y[0]                  Y[M-1]   Z[0]                  Z[M-1]
1000000000000000000000000   1101110010111001011100101
0100000000000000000000000   1110111001011100101110010
0010000000000000000000000   0111011100101110010111001
0001000000000000000000000   1011101110010111001011100
0000100000000000000000000   0101110111001011100101110
0000010000000000000000000   0010111011100101110010111
0000001000000000000000000   1001011101110010111001011
0000000100000000000000000   1100101110111001011100101
0000000010000000000000000   1110010111011100101110010
0000000001000000000000000   0111001011101110010111001
0000000000100000000000000   1011100101110111001011100
0000000000010000000000000   0101110010111011100101110
0000000000001000000000000   0010111001011101110010111
0000000000000100000000000   1001011100101110111001011
0000000000000010000000000   1100101110010111011100101
0000000000000001000000000   1110010111001011101110010
0000000000000000100000000   0111001011100101110111001
0000000000000000010000000   1011100101110010111011100
0000000000000000001000000   0101110010111001011101110
0000000000000000000100000   0010111001011100101110111
0000000000000000000010000   1001011100101110010111011
0000000000000000000001000   1100101110010111001011101
0000000000000000000000100   1110010111001011100101110
0000000000000000000000010   0111001011100101110010111
0000000000000000000000001   1011100101110010111001011
```

【図9】

Y[0] … Y[M-1]	Z[0] … Z[M-1]
1011000000000000000000000	0001000000000000000000000
0101100000000000000000000	0000100000000000000000000
0010110000000000000000000	0000010000000000000000000
0001011000000000000000000	0000001000000000000000000
0000101100000000000000000	0000000100000000000000000
0000010110000000000000000	0000000010000000000000000
0000001011000000000000000	0000000001000000000000000
0000000101100000000000000	0000000000100000000000000
0000000010110000000000000	0000000000010000000000000
0000000001011000000000000	0000000000001000000000000
0000000000101100000000000	0000000000000100000000000
0000000000010110000000000	0000000000000010000000000
0000000000001011000000000	0000000000000001000000000
0000000000000101100000000	0000000000000000100000000
0000000000000010110000000	0000000000000000010000000
0000000000000001011000000	0000000000000000001000000
0000000000000000101100000	0000000000000000000100000
0000000000000000010110000	0000000000000000000010000
0000000000000000001011000	0000000000000000000001000
0000000000000000000101100	0000000000000000000000100
0000000000000000000010110	0000000000000000000000010
0000000000000000000001011	0000000000000000000000001
0000000000000000000000100	1110010111001011100101110
0000000000000000000000010	0111001011100101110010111
0000000000000000000000001	1011100101110010111001011